Problem 1

Problem Description

In this problem you will create your own neural network to fit a function with two input features x_0 and x_1, and predict the output, y. The structure of your neural network is up to you, but you must describe the structure of your network, training parameters, and report an MSE for your fitted model on the provided data.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Visualization of provided data
- Visualization of trained model with provided data
- Trained model MSE
- Discussion of model structure and training parameters

Imports and Utility Functions:

In [10]:
```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch import optim

def dataGen():
    # Set random seed so generated random numbers are always the same
    gen = np.random.RandomState(0)
    # Generate x0 and x1
    x = 2*(gen.rand(200,2)-0.5)
    # Generate y with x0^2 - 0.2*x1^4 + x0*x1 + noise
    y = x[:,0]**2 - 0.2*x[:,1]**4 + x[:,0]*x[:,1] + 0.4*(gen.rand(len(x))-0.5)
```

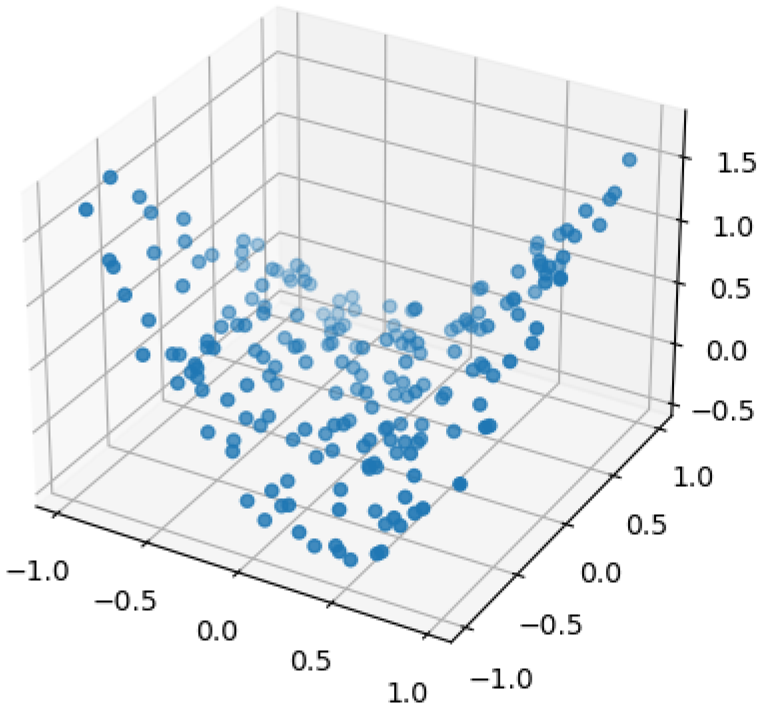
```
    return x, y

def visualizeModel(model):
    # Get data
    x, y = dataGen()
    # Number of data points in meshgrid
    n = 25
    # Set up evaluation grid
    x0 = torch.linspace(min(x[:,0]),max(x[:,0]),n)
    x1 = torch.linspace(min(x[:,1]),max(x[:,1]),n)
    X0, X1 = torch.meshgrid(x0, x1, indexing = 'ij')
    Xgrid = torch.vstack((X0.flatten(),X1.flatten())).T
    Ypred = model(Xgrid).reshape(n,n)
    # 3D plot
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    # Plot data
    ax.scatter(x[:,0],x[:,1],y, c = y, cmap = 'viridis')
    # Plot model
    ax.plot_surface(X0.detach().numpy(),X1.detach().numpy(),Ypred.detach().numpy(), color = 'gray', alpha = 0.25)
    ax.plot_wireframe(X0.detach().numpy(),X1.detach().numpy(),Ypred.detach().numpy(),color = 'black', alpha = 0.25)
    ax.set_xlabel('$x_0$')
    ax.set_ylabel('$x_1$')
    ax.set_zlabel('$y$')
    plt.show()
```

Generate and visualize the data

Use the `dataGen()` function to generate the x and y data, then visualize with a 3D scatter plot.

In [11]:

```
# YOUR CODE GOES HERE
x, y = dataGen()
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y)
plt.show()
```



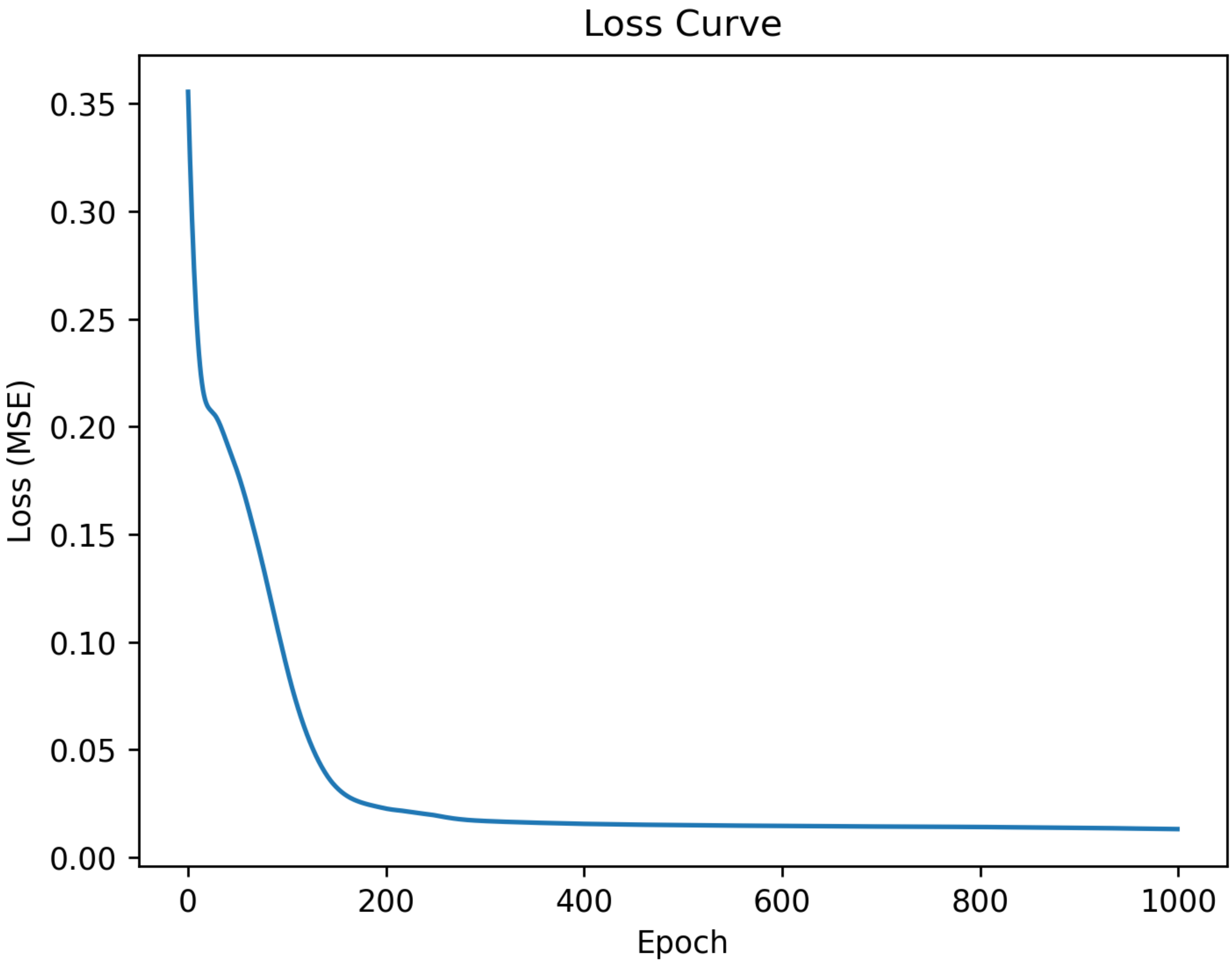
Create and train a neural network using PyTorch

Choice of structure and training parameters are entirely up to you, however you will need to provide reasoning for your choices. An MSE smaller than 0.02 is reasonable.

In [25]:
```
# YOUR CODE GOES HERE



x = torch.Tensor(x)
y = torch.Tensor(y)
y = y.reshape(-1,1)
loss_curve =[]


class Net_2_layer(nn.Module):
    def __init__(self, N_hidden= 4, N_in=2, N_out=1, activation = F.relu):
        super().__init__()
```

```
        self.lin1 = nn.Linear(N_in, N_hidden)
        self.lin2 = nn.Linear(N_hidden, N_out)
        self.act = activation

    def forward(self,x):
        x = self.lin1(x)
        x = self.act(x)
        x = self.lin2(x)

        return x

model = Net_2_layer(N_hidden = 8, N_in = 2, N_out = 1, activation = F.relu)

lr = 0.005
epochs = 1000
loss_fcn = F.mse_loss

opt = optim.Adam(params = model.parameters(), lr=lr)

for epoch in range(epochs):
    out = model(x)

    loss = loss_fcn(out,y)

    loss_curve.append(loss.item())
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAverage loss: {loss.item()}")

    opt.zero_grad()
    loss.backward()
    opt.step()

plt.figure(dpi=250)
plt.plot(loss_curve)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Loss Curve')
plt.show()
```

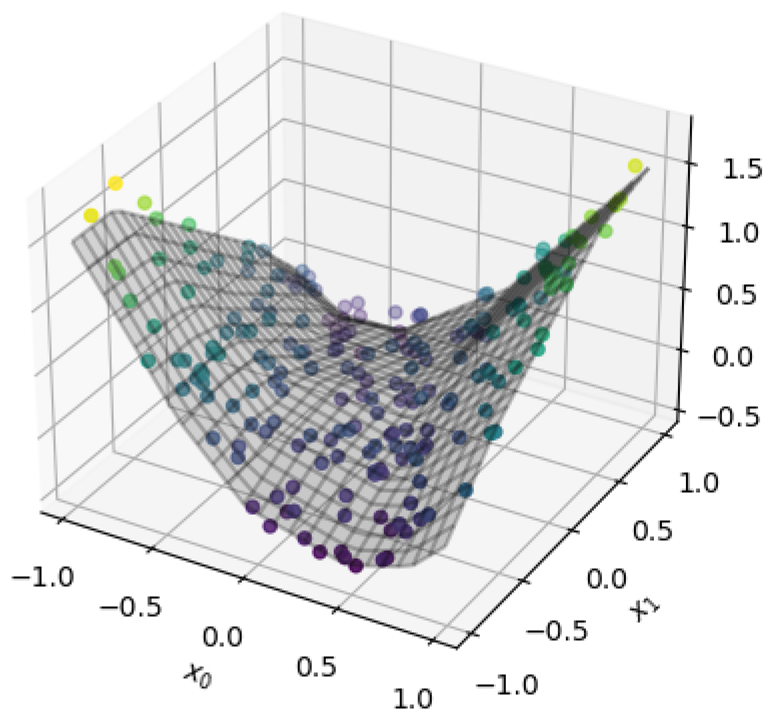
```
Epoch 0 of 1000...      Average loss: 0.3554549515247345
Epoch 40 of 1000...     Average loss: 0.19141222536563873
Epoch 80 of 1000...     Average loss: 0.12670733034610748
Epoch 120 of 1000...    Average loss: 0.057165950536727905
Epoch 160 of 1000...    Average loss: 0.028676504269242287
Epoch 200 of 1000...    Average loss: 0.02264014631509781
Epoch 240 of 1000...    Average loss: 0.020118260756134987
Epoch 280 of 1000...    Average loss: 0.017450470477342606
Epoch 320 of 1000...    Average loss: 0.016507450491189957
Epoch 360 of 1000...    Average loss: 0.015976298600435257
Epoch 400 of 1000...    Average loss: 0.015547108836472034
Epoch 440 of 1000...    Average loss: 0.015252158977091312
Epoch 480 of 1000...    Average loss: 0.015041529200971127
Epoch 520 of 1000...    Average loss: 0.014879926107823849
Epoch 560 of 1000...    Average loss: 0.014711434952914715
Epoch 600 of 1000...    Average loss: 0.014565806835889816
Epoch 640 of 1000...    Average loss: 0.014451917260885239
Epoch 680 of 1000...    Average loss: 0.01435183547437191
Epoch 720 of 1000...    Average loss: 0.014256665483117104
Epoch 760 of 1000...    Average loss: 0.01417855266481638
Epoch 800 of 1000...    Average loss: 0.01409836020320654
Epoch 840 of 1000...    Average loss: 0.013957083225250244
Epoch 880 of 1000...    Average loss: 0.0137586435303092
Epoch 920 of 1000...    Average loss: 0.013619357720017433
Epoch 960 of 1000...    Average loss: 0.013349588960409164
```



Visualize your trained model

Use the provided `visualizeModel()` function by passing in your trained model to see your models predicted function compared to the provided data

In [26]:
```
# YOUR CODE GOES HERE
visualizeModel(model)
```



Discussion

Report the MSE of your trained model on the generated data. Discuss the structure of your network, including the number and size of hidden layers, choice of activation function, loss function, optimizer, learning rate, number of training epochs.

YOUR ANSWER GOES HERE

The MSE is found to be 0.013.

There was two input layer(N_in) and one output layer(N_out) and only one hidden layer was used with 8 layers of preceptron. Smaller preceptron layers were tried i.e., 2,4 before using 8 but 8 gave us the MSE less tahn 0.02.

Along with that the activation function used was Relu. Because it leads to a fast convergence i.e., it is faster to compute and it also helps in overcoming vanishing gradients.

The loss function used was MSE.loss because it is not a classification problem and for regression problems MSE.loss is used. As it is a regression function there was no activation used in the final layer in the forward function.

The optimizer used was Adam as it converges faster than ay other optimizer.

As a smaller learning rate leads to a better fit of the curve. So, the learning rate used was 0.005. A lower and higher learning rates were used too but this provided teh results needed for the problem.

The number of iterations or number of training epochs was kept at 1000. This was the best to use as smaller number of training epoch provided a higher MSE and a larger number of training epochs provided an overfitting curve.